Kevoree : une approche model@runtime pour les systèmes ubiquitaires

François Fouquet, IRISA, Triskell, Université de Rennes 1
Erwan Daubert, INRIA, Triskell, Université de Rennes 1
Olivier Barais, IRISA, Triskell, Université de Rennes 1
Noel Plouzeau, IRISA, Triskell, Université de Rennes 1
Johann Bourcier, IRISA, Triskell, Université de Rennes 1
Jean-Emile Dartois, INRIA, Triskell, Université de Rennes 1
Arnaud Blouin, IRISA, Triskell, Université de Rennes 1

Résumé : *La prolifération d'équipements autour de la mouvance de l'Internet des objets amène une grande diversité des usages et des plates-formes d'exécutions. Ces usages exploitent de manière croissante le caractère distribué et autonomique de ces équipements amenant une vraie complexité de développement. La réflexion nécessaire à ces usages afin de profiter de ces équipements de manière optimale devient alors une préoccupation majeure. L'approche Models@Runtime vise justement à encadrer et étendre la gestion de cette couche de réflexion distribuée. Cet article présente Kevoree : une approche outillée autour de ce paradigme pour répondre à des enjeux d'ingénierie logicielle telle que la construction d'un système tactique de terrain pour les opérations d'urgence. Ce système tactique est développé en collaboration avec le Service Départemental d'Incendie et de Secours dans le cadre du projet DAUM. Ce projet tire partie de l'approche à composants de Kevoree pour mettre en œuvre un système distribué adaptatif composé de nœuds hétérogènes (Java et capteurs embarqués).*

Mots-clés : *Système ubiquitaire, adaptation dynamique, CBSE, models@runtime.*

1 Introduction

La multiplication des équipements mobiles a permis une diversification de leurs usages pour tendre vers l'émergence de nouvelles applications ubiquitaires. Cette évolution complexifie le développement des logiciels fondés sur ces nouveaux usages en ajoutant le problème de la prises en compte des propriétés non-fonctionnelles comme, par exemple, la gestion de la distribution, la gestion de la concurrence, la capacité à s'adapter en fonction du contexte et la gestion de communications non fiables.

Il existe de nombreux types de systèmes ubiquitaires. Dans cet article nous nous intéressons à des systèmes dotés des caractéristiques suivantes :

- grande hétérogénéité des types de nœuds, en terme de puissance de calcul, de mémoire et de capacité réseau ;
- rapide variation de la taille de l'application, en terme de nombre de nœuds ;
- emploi de réseaux d'interconnexion « incertains ».

Dans ce cadre, le projet Kevoree est une plate-forme construite en suivant les principes du models@runtime [11] en le confrontant aux problématiques des systèmes ubiquitaires (distribution, concurrence, dynamicité, hétérogénéité et incertitude). L'objectif de cet article est de présenter les propriétés offertes par la plate-forme Kevoree. Celles-ci concernent notament l'adaptatbilité en fonction du contexte, de l'hétérogénéité des plates-formes d'exécution [9] et de la dissémination des politiques de reconfiguration dans un contexte de connectivité non fiable [8] en s'appuyant sur un cas concrêt de système d'information tactique. Ce système est développé en coopération avec des sapeurs-pompiers afin de valider l'infrastructure logicielle présentée. Ce système, construit dans le cadre d'un projet nommé DAUM, est une application d'aide à la décision temps réel et de terrain. Il est composé principalement d'une application de travail coopératif fonctionnant sur des tablettes tactiles, de capteurs intégrés à l'équipement individuels de sapeurs-pompiers, et d'un système de supervision réparti constitué de nœuds de calcul et de donnée embarqués dans les véhicules des intervenants.

Le système gère de façon transparente les besoins d'adaptation suivants :

- apport massif de capteurs suite à l'arrivée de renforts ;
- réorganisation des groupes tactiques, en termes d'attribution de secteur géographique et de domaine de tâches ;
- montée en puissance rapide des moyens sapeurs-pompiers, impliquant une allocation et une reconfiguration rapide et automatique de l'infrastructure de calcul et de communication sur le terrain.

La papier vise à présenter la gestion de ces contraintes non fonctionnelles à l'aide de la plate-forme Kevoree.

La reste du papier est structuré de la manière suivante. Nous présentons d'abord le contexte du projet DAUM, les exigences non-fonctionnelles d'un tel système et l'architecture retenue pour l'application. Une deuxième section résume les concepts principaux et les fonctionnalités offertes par la plate-forme Kevoree. La section suivante détaille le scénario proposé pour l'évaluation de Kevoree. Finalement une rapide conclusion détaille les travaux en cours autour de ce projet.

2 Le système DAUM

Cette section décrit l'architecture fonctionnelle de projet DAUM.

2.1 Fonctions de DAUM

Le système tactique de terrain DAUM doit assurer les grandes fonctions suivantes.

Tableau électronique coopératif de terrain

Lors d'une intervention, chaque responsable d'un secteur et chaque officier impliqué dans la chaîne de commandement doit disposer sur le terrain d'une tablette tactile ou d'un tableau tactile présentant la situation tactique. Ce support tactile doit également inclure différents outils conceptuels définis au niveau national pour les sapeurs-pompiers pour la gestion opérationnelle du commandement. Le système se comporte comme une application de travail coopératif assisté par ordinateur (TCAO) de terrain, avec mise à jour en temps réel par tous les intervenants de l'intervention. L'interconnexion entre les tablettes se fait de façon transparente pour les utilisateurs d'une intervention, en utilisant les différentes liaisons sans-fil disponibles.

Aide à la sécurité individuelle et collective

Chaque sapeur-pompier est doté d'un système de supervision de constantes physiques(rythme cardiaque, température de sol, etc.), intégré dans l'équipement de protection individuelle. Lors d'une intervention, les systèmes individuels sont configurés pour observer les paramètres pertinents, en fonction du type de l'intervention (par exemple longue reconnaissance, feu d'aire naturelle, risque chimique, etc.) et du rôle affecté à chaque sapeur-pompier (attaque d'un foyer en espace semi-clos, alimentation en eau, etc.). Chaque système individuel est alors à même d'avertir son porteur de l'évolution d'une situation physique externe (risque d'embrasement généralisé éclair, atmosphère explosive, etc.) ou interne (rythme cardiaque élevé, température corporelle élevée, etc.). Les informations sont également transmises au système de terrain et analysées de façon automatique pour attirer l'attention des utilisateurs du système tactique évoqué plus haut et également pour avoir une vue plus précise de la situation tactique.

Interconnexion avec le système de gestion opérationnel central

Les interventions sapeur-pompier sont gérées par un centre opérationnel fixe, appelé CODIS. Le système DAUM doit s'intégrer à ce système opérationnel, pour permettre une gestion directe des demandes de moyens supplémentaires.

2.2 Architecture distribué de DAUM

Le projet DAUM est déployé sur un environnement massivement distribué. Il est composé de trois types de nœuds, les nœud de type BS (basse consommation), C (commandement) et S (soutien). Un nœud dans la terminologie DAUM correspond à une unité de calcul autonome capable d'analyser et de transmettre en temps réel des données. Le réseau résultant de l'interconnexion de ces nœuds permet d'interconnecter les sapeurs-pompiers de proche en proche, formant un maillage de nœuds indépendants les uns des autres. Les données de la situation tactique sont distribuées sur l'ensemble des nœuds participant à l'intervention.

Les nœuds BS

Les nœuds de type BS sont implémentés sous la forme de micro-contrôleurs basse puissance, avec une fréquence d'horloge de quelques mégahertz et quelques kiloctets de mémoire vive. Cela offre de nombreux avantages, en particulier leur très faible coût d'achat ainsi que leur importante autonomie électrique. Ces nœuds sont intégrés dans ou sur les vêtements des sapeurs-pompiers. Ils permettent grâce à un ensemble de capteurs de collecter grâce à différents montages électronique, tels qu'ADC (Analog Digital Converter), I2C (Inter Integrated Circuit), UART (Universal Asynchronous Receiver Transmitter), des informations sur l'environnement et l'état physiologique des pompiers (figure 1). Ils sont à l'épicentre d'une intervention, car bien placés pour détecter des risques, tels que des gaz toxiques ou l'hyperthermie, ainsi que pour prévenir et alerter le porteur puis la chaîne de commandement.

L'évaluation des risques est réalisée en temps réel grâce à un moteur de logique floue embarqué. Les règles sont formulées par un expert du domaine en langage naturel grâce à un langage dédié (DSL – Domain Specific Language). Les paramètres, tels que la période de mise à jour des capteurs et les partitions de la logique floue, peuvent être modifiés dynamiquement pendant l'intervention.

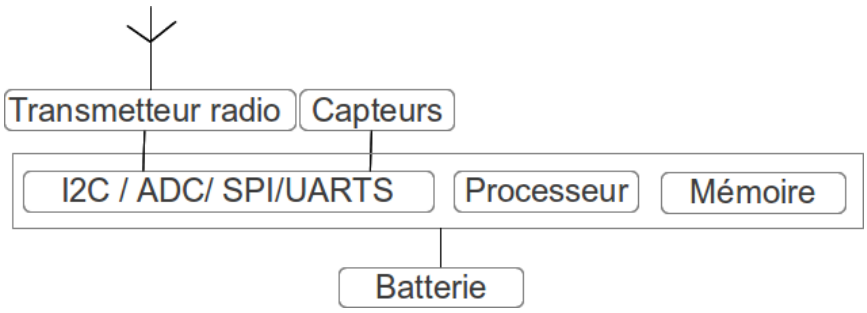

FIG. 1: Architecture physique d'un nœud BS

Les nœuds S

Les nœuds de type S fonctionnent sur des systèmes embarqués disposant d'un processeur basse consommation (par exemple, type ARM), l'horloge variant d'une centaine de mégahertz à quelques gigahertz. Le coût d'un nœud peut varier de plusieurs centaines d'euros à une vingtaine d'euros pour un nœud construit avec la carte Raspberry Pi. Les nœuds S sont embarqués dans les véhicules des sapeurs-pompiers, qui fournissent l'énergie électrique. Contrairement aux nœuds BS, ils disposent d'un système d'exploitation complet et de plusieurs gigaoctets de mémoire flash. Cette puissance de calcul permet d'exécuter des algorithmes complexes d'aide à la décision, comme des algorithmes génétiques. Cet espace mémoire est aussi utilisé pour regrouper les données tactiques de l'intervention émises par les nœuds BS, mais aussi les données volumineuses comme les données cartographiques. Ces nœuds sont associés à des émetteurs radio d'une portée de plusieurs kilomètres.

Les nœuds C

Les nœuds de type C fonctionnent sur des tablettes tactiles ou des tableaux de plus grande taille. Chaque tablette héberge une application spécialisée d'aide à la décision. Cette interface permet à un sapeur-pompier d'appliquer la méthode de raisonnement tactique en vigueur chez les sapeurs-pompiers français. Elle met à disposition les outils normalisés associés (ordre graphique, SITAC,

tableau des moyens, tableau des messages, etc.). Pour une intervention en cours, l'ensemble des données disponibles (plans, relevé de la situation, des objectifs, des tactiques, des moyens sur place ou demandés, etc.) sont accessibles aux intervenants sur leur tablette. L'application se comporte comme un éditeur spécialisé multi-utilisateur temps réel. La mise en œuvre de cette application s'appuie sur l'infrastructure de composants dynamiques Kevoree (cf. section 3). Les différents éléments de l'application sont mis en œuvre par des composants Kevoree. L'application bénéficie ainsi des propriétés de reconfiguration dynamique et de gestion de la distribution offertes par le modèle Kevoree. Les nœuds de type BS, S et C sont interconnectés en réseau de type Mesh (figure 2).

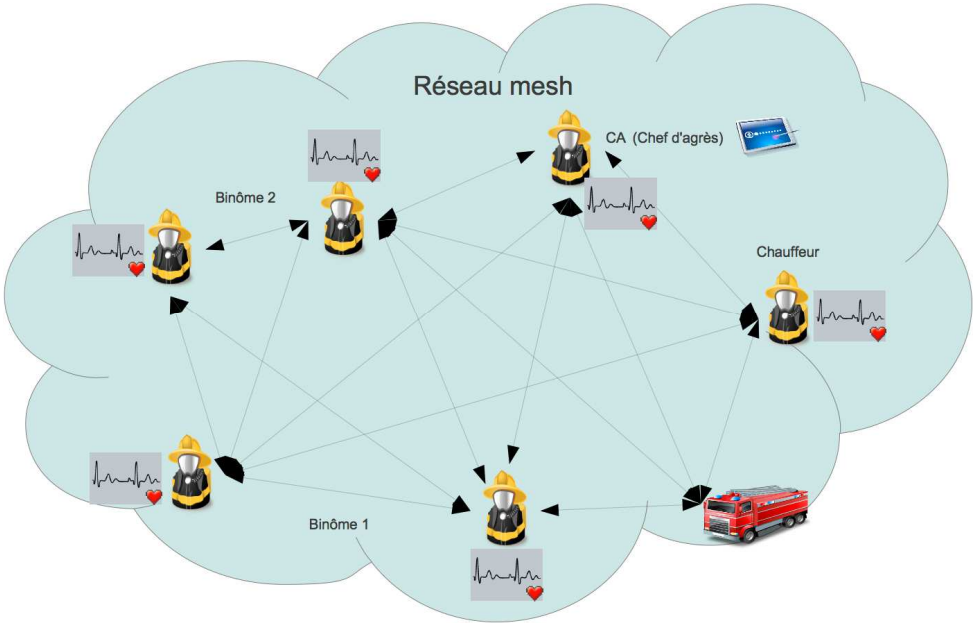

FIG. 2: Vue globale de l'architecture

3 Kevoree

Le projet Kevoree [1], construit en suivant le paradigme de présence des modèles à l'exécution (Models@runtime) [11], est un cadre de développement d'applications à base de composants pour gérer la conception des systèmes adaptatifs distribués. Cette modélisation des architectures logicielles vise donc à proposer un modèle abstrait au travers duquel il est possible de manipuler les concepts importants d'une application tels que les composants, les services offerts par ces composants, les canaux de communications ou encore les groupes de synchronisation. Le modèle de composant de Kevoree couvre également les concepts d'infrastructure exécutant l'application. Celle-ci est alors modélisée sous la forme de nœuds logiques représentant soit des nœuds de calcul physique, soit des nœuds virtuels. À ces nœuds est adjoint un modèle de topologie d'organisation réseau.

3.1 Models@Runtime

L'approche Models@Runtime consiste à maintenir à l'exécution une représentation modèle du système en cours d'exécution. Ce modèle abstrait a la capacité de se désynchroniser et resynchroniser à la demande par rapport à la plate-forme concrète. L'ensemble des adaptations du système se font alors par manipulation offline de ce modèle, puis par resynchronisation. Lors des modifications du modèle, cette approche permet de s'abstraire des contraintes liées à l'application des adaptations (par exemple l'ordonnancement des adaptations). La figure 3 présente une vue générale du processus de cette approche. Ainsi, les adaptations prennent la forme d'un nouveau

1 http://kevoree.org

modèle (nommé modèle cible) à appliquer sur le système. Ce modèle cible est préalablement validé, afin de vérifier sa conformité vis-à-vis des contraintes du système. Le modèle cible est ensuite comparé au modèle courant représentant le système en cours de fonctionnement. Cette comparaison produit un ensemble de primitives abstraites permettant de construire le modèle cible à partir du modèle représentant le système en cours d'exécution. Après les avoir ordonnées, le moteur d'adaptation exécute des actions d'adaptations concrètes liées à ces primitives. Cette exécution se fait dans un mode transactionnel. Si l'une des actions échoue, le moteur d'adaptation annule toutes les modifications pour revenir à un état stable et cohérent. Si l'ensemble des actions s'exécute correctement, le modèle cible devient alors le modèle courant. Le point clé de cette approche est donc d'assurer la cohérence entre le modèle courant et le système en cours d'exécution. Le projet Kevoree exploite cette approche dans un contexte distribué imposant donc une synchronisation des différents modèles courants.

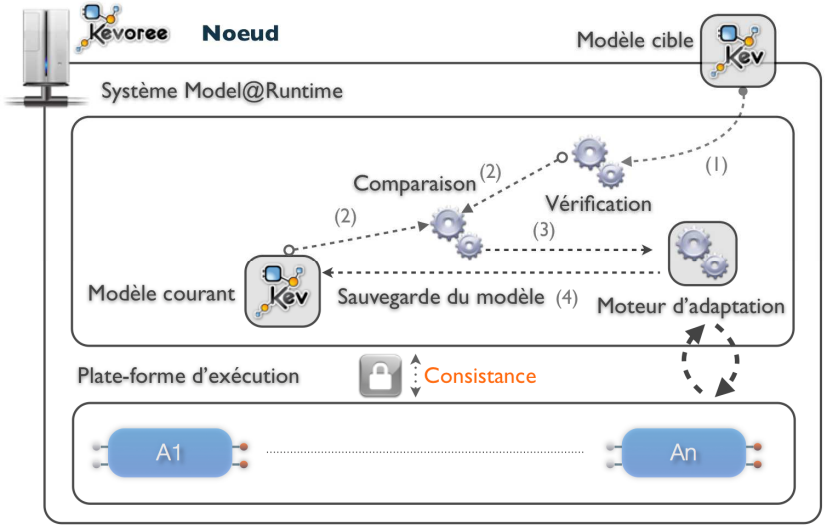

FIG. 3: Vue globale du processus Models@Runtime

3.2 DSLs d'architecture

Le projet Kevoree inclut un ensemble d'outils pour manipuler les différents concepts proposés. Ces outils sont développés sous la forme de langages dédiés (DSL) graphiques ou textuels permettant de définir une configuration d'un système.

L'un d'eux, nommé KevScript, inspiré par d'autres projets tel que FScript [6], permet de faciliter la modification du modèle d'architecture. Suivant le paradigme models@runtime, le projet Kevoree exploite les modèles d'architectures comme une couche de réflexion désynchronisée. L'exécution du moteur KevScript se fait au niveau du modèle sans agir directement sur l'architecture, permettant ainsi de s'abstraire des effets des modifications et de leurs relations. Ce type d'approche est exploité notamment comme cible de compilation par les moteurs de raisonnement devant adapter un système ubiquitaires en cours de fonctionnement.

La dissémination des versions de modèle représentant les changements d'architecture se fait donc de manière asynchrone sur demande explicite des processus de mise à jour. Ce type de propagation est particulièrement adapté pour des réseaux maillés fortement contraints [8].

3.3 Concepts de Kevoree

La modélisation d'un système à base de composants est particulièrement appropriée pour les systèmes adaptatifs [12]. Dans cette section, nous décrivons les différents paradigmes de l'approche Kevoree qui permettent l'adaptation dynamique de logiciels et de l'infrastructure d'hébergement.

Patron de conception Type/Instance

Pour mettre à jour un système de façon continue, il est important de pouvoir différencier les types représentant les fonctionnalités disponibles, des instances représentant l'usage localisé de ces

fonctionnalités. C'est une nécessité à la fois pour raisonner sur la substituabilité des fonctionnalités mais également pour connaître les instances concernées par une mise à jour de type.

L'ensemble des concepts de Kevoree suit donc un patron de conception type/instance [10] qui, à la manière de la programmation objet, sépare les définitions (par exemple Classe) des instances (par exemple Objet).

De ce fait, les adaptations concernant les définitions de types se font de manière désynchronisée du modèle d'exécution. Elles sont ensuite appliquées sur celui-ci après validation du calcul des effets sur les instances. Les adaptations concernant les instances (paramètres, localisation) sont directement effectuées sur le modèle d'exécution. Dans les deux cas la propagation du nouveau modèle d'exécution s'effectue systématiquement de manière asynchrone. La gestion et résolution des éventuels conflicts pouvant apparaitre est délégué aux entités groupes détaillés par la suite.

Cycle de vie

Afin de permettre leur usage dans un système adaptatif, l'ensemble des concepts de Kevoree identifiés sous le terme définition de type doivent définir un cycle de vie. Ce cycle de vie est identifié par un processus à effectuer au démarrage, à l'arrêt et à la mise à jour d'une instance. Les définitions de types définissent également un type de dictionnaire modélisant les paramètres disponibles pour les instances de ce type. De manière analogue, les instances possèdent donc un ensemble de paramètres respectant le dictionnaire de leur type. La mise à jour d'une instance correspond à une évolution de ces paramètres (nommée mise à jour paramétrique).

Composants

Suivant une définition communément admise, les composants Kevoree définissent un contrat d'interface [3]. Ce contrat définit un ensemble de fonctionnalités fournies, requises et identifiés de manière unique par un port d'interface. Un composant définit donc une fonctionnalité offerte au système. Un composant est défini comme substituable à un autre s'il possède au moins l'ensemble des ports du même type. Cette caractéristique permet de reconfigurer les applications au design-time et au runtime tout en maintenant les fonctionnalités requises.

Canaux de communication (Channels)

En plus des composants, une application définit aussi les relations entre ses composants afin de les faire collaborer. Ces relations sont représentées sous forme de canaux (channels) qui encapsulent les sémantiques de communication entre composants. Cette sémantique devient particulièrement complexe dans le cas de relation 1-N ; on peut alors définir des modèles de communication transactionnels ou de type broadcast.

Nœuds

Un système distribué est caractérisé par un ensemble de nœuds de calcul. Chaque nœud peut aussi bien héberger un ensemble de composants et de channels, que d'autres nœuds (afin de faciliter la définition de couche de virtualisation). Ainsi les nœuds sont organisés de manière hiérarchique, le parent étant responsable du cycle de vie des nœuds fils. Un nœud est donc un conteneur qui fournit un niveau d'isolation et qui est responsable localement de la synchronisation entre son modèle d'architecture et le système qui s'exécute. Cette responsabilité est représentée par les capacités d'adaptations du nœud. Celles-ci sont définies et mise en œuvre par des primitives d'adaptations concrètes permettant d'effectuer les différentes étapes de la migration entre deux configurations (deux modèles). L'hétérogénéité des plates-formes d'exécution est donc gérée via ces primitives d'adaptation, l'ajout d'une nouvelle plate-forme consistant à définir un ensemble d'actions concrètes. Ainsi le projet Kevoree définit d'ores et déjà plusieurs types de nœud, aussi bien pour des architectures embarqués telles que des microcontrolleurs [9] (par exemple Arduino[2]) ou des plate-

2 http://www.arduino.cc/

formes Android[3] que pour des plate-formes plus lourdes telles que des nœuds JavaSE[4] ou de la virtualisation avec FreeBSDJail[5].

Groupes

Un groupe est dédié à la synchronisation de la représentation par modèle d'un ensemble de nœuds. Cette synchronisation définit une portée spécifiée par un protocole de synchronisation mais également un protocole de communication entre un ensemble de nœuds. De la même façon que les channels sont utilisés pour définir la sémantique de communication entre les composants, les groupes sont utilisés pour définir la sémantique de communication pour la dissémination du models@runtime entre les nœuds. La cohérence de l'ensemble du système est donc assurée par les groupes de synchronisation. La figure 4 illustre l'organisation des nœuds autour d'un groupe.

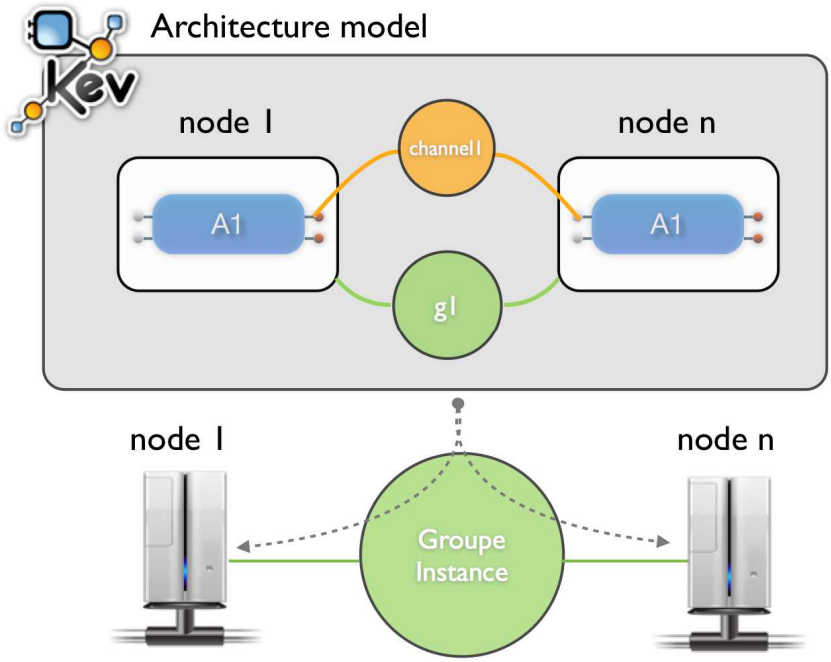

FIG. 4: Exemple d'usage de groupe

4 Cas d'étude

Cette section présente l'un des cas d'étude issu du projet DAUM, concernant la sécurité des sapeurs-pompiers dans le cadre d'intervention sur des incendies.

L'un des dispositifs de sécurité intégré dans l'équipement d'un sapeur-pompier est la balise homme-mort. Cet appareil surveille l'activité de son porteur et l'alerte en cas de danger en transmettant cette alerte au système DAUM. Plus précisément, la balise homme-mort surveille les déplacements du pompier et donne l'alerte si celui-ci ne bouge plus depuis un certain temps. L'alerte est donnée par une alarme lumineuse et sonore afin de faciliter la détection du pompier dans des environnements difficiles (absence de visibilité due aux fumées de l'incendie par exemple). Un stade de pré-alerte est aussi défini pour notifier le pompier qu'il est immobile depuis un certain temps et ainsi lui permettre de faire un mouvement prouvant qu'il n'est pas en difficulté. À noter que les balises homme-mort quotidiennement utilisées par les sapeurs-pompiers ne gèrent que l'absence de mouvement et ne sont pas connectées en réseau.

Une balise homme-mort DAUM se compose d'une LED pour l'alarme lumineuse, d'un buzzeur pour l'alarme sonore, d'un accéléromètre pour détecter les mouvements et enfin d'un moteur de

3 http://www.android.com/
4 http://www.oracle.com/technetwork/java
5 http://www.freebsd.org/handbook/jails.html

raisonnement chargé d'effectuer la surveillance des mouvements à partir des données fournies par l'accéléromètre et d'actionner l'alarme. La figure 5 représente le schéma d'une balise homme-mort développée avec Kevoree.

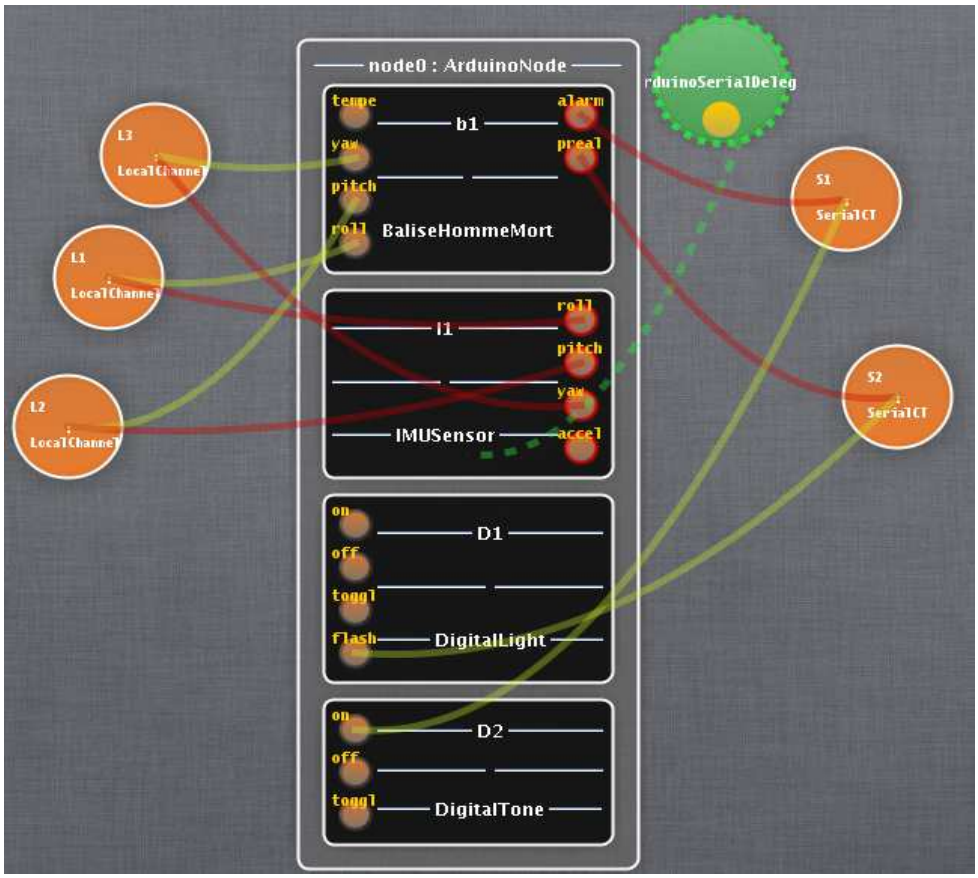

FIG. 5: Modèle Kevoree de la balise homme-mort

La détection de risque d'embrasement généralisé est un système complémentaire à la balise homme-mort. L'embrasement généralisé est un phénomène thermique pouvant se produire dans le cadre d'un incendie de bâtiment lorsque la température est telle que l'ensemble de la pièce peut s'enflammer violemment en quelques secondes. L'une des solutions de détection de risque d'embrasement généralisé consiste à mesurer la température de la pièce. Ce dispositif se compose d'un thermomètre laser, d'une LED pour informer le pompier du niveau de dangerosité et d'un dispositif mesurant la température et mettant à jour le niveau d'alerte. La figure 6 représente le schéma de d'un détecteur d'embrasements généralisés.

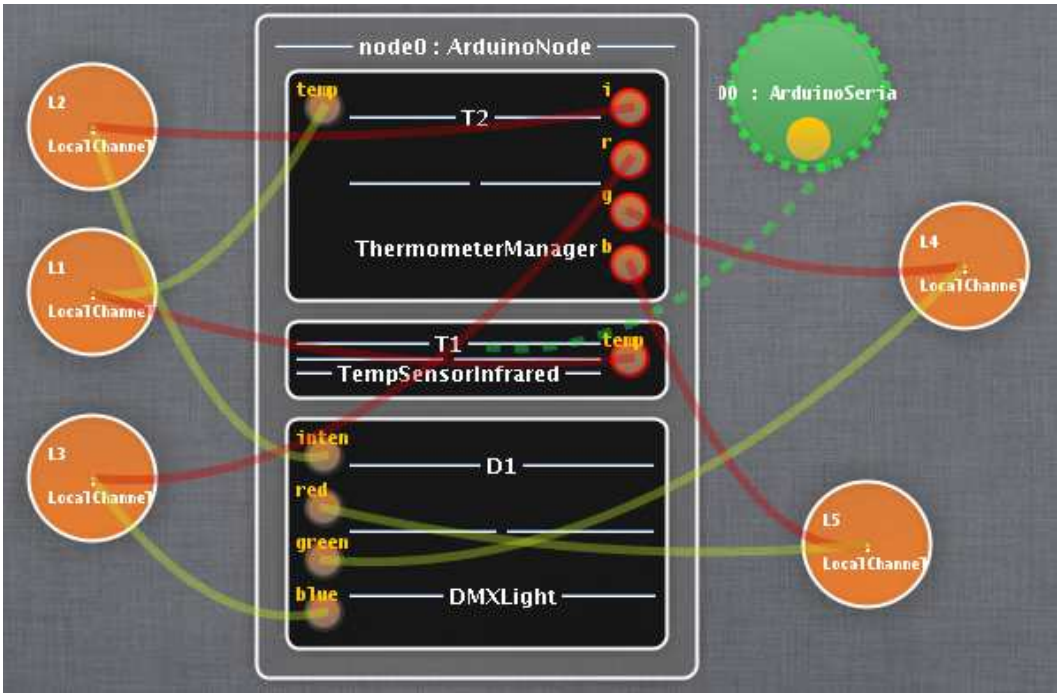

FIG. 6: Modèle Kevoree pour la détection de la température

Les systèmes de sécurité dont dispose le sapeur-pompier dépendent de la situation. Par exemple, dans le cadre d'une intervention sur un accident de la route, il n'est pas nécessaire de détecter le risque d'embrasement généralisé puisqu'il n'y a pas d'incendie. Pour autant, il n'est pas évident pour les sapeurs-pompiers de modifier, avant chaque intervention, le dispositif électronique inclus dans son matériel (par exemple dans la doublure de sa veste) en fonction de l'intervention. C'est pourquoi il est nécessaire d'intégrer dans la veste d'un sapeur-pompier la totalité des capteurs nécessaires aux différents systèmes de sécurité. Ces capteurs sont automatiquement configurés en fonction des informations et des choix pris par le centre de commandement (définition du type d'intervention, niveau de risque dû à des produits toxiques, etc).

De plus l'interconnexion des équipements permet de partager les données fournies par l'ensemble des personnes engagées sur une intervention. Ce partage d'information peut se révéler utile voire primordial dans le cadre de la prise de décision par la structure de commandement mais aussi dans le cadre de la sécurité individuelle et collective des agents. Par exemple, l'utilisation de l'ensemble des données de température calculée par les thermomètres de chacun des agents peut permettre une évaluation des flux thermiques plus précise et ainsi affiner le calcul du risque d'embrasement généralisé. La figure 7 représente une partie du système des sapeur-pompiers avec deux agents sur lesquels sont configurés les systèmes de détection de risque d'embrasement généralisé et un nœud de commandement sur lequel l'officier responsable du secteur peut suivre l'évolution des données et agir en fonction de celles-ci.

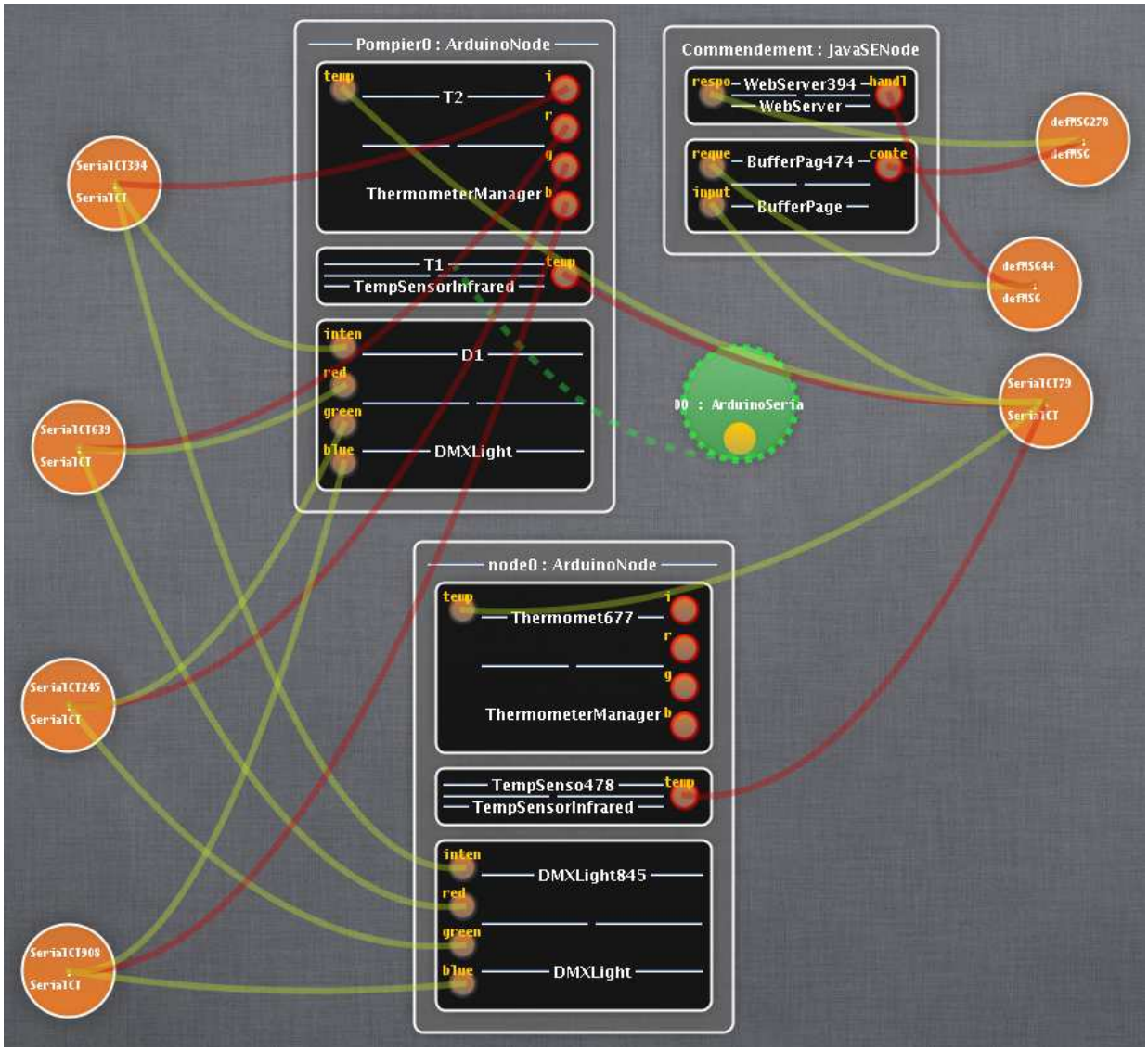

FIG. 7: Modèle Kevoree du système distribué

5 Travaux Connexes

Il existe de nombreux cadres de développement permettant de développer des applications ubiquitaires modulaires. Parmi celles-ci, Frascati [12] fournit un support pour la reconfiguration dynamique d'applications modulaires construites autour du standard SCA. De même, iPojo [7] est un outil du projet Apache Felix[6] permettant de simplifier le développement de services OSGi à l'aide d'un modèle de composant. Il s'occupe de la gestion des dépendances, de l'injection de références dans les variables des composants, etc. Kevoree se distingue principalement de ces deux projets par l'introduction du concept de groupe et de nœud dans le modèle de composant permettant de gérer l'hétérogénéité des plates-formes de reconfiguration et la dissémination des politiques de reconfiguration[8, 9]. Parmi les autres approches, DiaSuite [5] permet d'aider le développement de ces applications ubiquitaires. DiaSuite est une approche outillée s'appuyant sur les langages de conception dédiés, offrant un support à chaque étape du développement : conception, implémentation, test et déploiement. DiaSuite permet d'assurer la fiabilité et la sécurité de ces systèmes en facilitant l'intégration des préoccupations non fonctionnelles dès les premières étapes du processus de développement. Si DiaSuite offre un meilleur support pour l'ensemble des phases du cycle de vie d'une application ubiquitaire, Kevoree travaille plus spécifiquement sur la prise en

6 http ://felix.apache.org/site/apache-felix-ipojo.html

compte de l'adaptabilité dans un contexte distribué avec nativement un support de la problématique de dissémination des politiques de reconfiguration.

L'adaptation dynamique des systèmes ubiquitaires doit également considérer l'adaptation de leurs parties interactives (interactions et interfaces homme-machine – IHM). Des travaux ont déjà été menés sur ce thème [4, 1, 13, 2]. En particulier, Avouac et al présentent une approche basée composant pour l'adaptation dynamique d'interactions multi-modales en environnement ubiquitaire [1]. Également, Blouin et al introduisent une approche pour l'adaptation dynamique des systèmes interactifs. Dans nos travaux futurs, cette approche sera combinée à Kevoree pour y intégrer la gestion des adaptations des IHM.

6 conclusion

Cet article présente une approche models@runtime (appelé Kevoree) pour faciliter le développement d'applications dans le contexte de l'informatique ubiquitaire distribuée. Kevoree est une plate-forme libre permettant à la fois la modélisation de l'architecture d'un système distribué hétérogène, tels que les systèmes ubiquitaires, et la gestion des évolutions au cours de leur fonctionnement.

La validation de notre plate-forme Kevoree s'appuie sur la mise en œuvre d'un ensemble de services visant à améliorer la sécurité de pompiers en intervention. Ces expérimentations ont été menées dans le cadre du projet DAUM. La validation menée dans cette étude a permis de mettre en avant la faisabilité de l'implantation de ce type de technologie sur des plates-formes embarquées très contraintes en capacité de calcul et espace mémoire.

Ces travaux ouvrent des perspectives de recherche concernant les processus de raisonnement pouvant tirer parti des capacités offertes par Kevoree pour gérer de façon autonomique les applications ubiquitaires. D'autre part, nous pensons que l'approche employée par la plate-forme Kevoree peut être utilisée aussi bien pour des plates-formes embarquées que pour des infrastructures conséquentes du type Cloud. La continuité de paradigme de programmation entre ces différentes entités offre de nouvelles perspectives à explorer concernant le développement d'applications.

Références

[1] P.-A. Avouac, P. Lalanda, and L. Nigay. Adaptable multimodal interfaces in pervasive environments. In Conference Proceedings of CCNC 2012, IEEE Consumer Communications and Networking Conference (14-17 January, USA). To appear, page 5, 2012.

[2] L. Balme, A. Demeure, N. Barralon, J. Coutaz, and G. Calvary. CAMELEON-RT : A software architecture reference model for distributed, migratable, and plastic user interfaces. In EUSAI, pages 291–302, 2004.

[3] A. Beugnard, J. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. Computer, 32(7) :38–45, 1999.

[4] A. Blouin, B. Morin, O. Beaudoux, G. Nain, P. Albers, and J.-M. Jézéquel. Combining aspect-oriented modeling with property-based reasoning to improve user interface adaptation. In EICS'11 : Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems, pages 85–94, 2011.

[5] D. Cassou, E. Balland, C. Consel, and J. Lawall. Leveraging Software Architectures to Guide and Verify the Development of Sense/Compute/Control Applications. In ICSE'11 : Proceedings of the 33rd International Conference on Software Engineering, pages 431–440, Honolulu, United States, 2011. ACM.

[6] P.-C. David, T. Ledoux, T. Coupaye, and M. Léger. FPath and FScript : Language support for navigation and reliable reconfiguration of Fractal architectures. annals of telecommunications, 2008.

[7] C. Escoffier and R. S. Hall. Dynamically adaptable applications with ipojo service components. In M. Lumpe and W. Vanderperren, editors, Software Composition, volume 4829 of Lecture Notes in Computer Science, pages 113–128. Springer, 2007.

[8] F. Fouquet, E. Daubert, N. Plouzeau, O. Barais, J. Bourcier, and J.-M. Jézéquel. Dissemination of reconfiguration policies on mesh networks. In DAIS 2012, Stockholm, Suède, June 2012.
[9] F. Fouquet, B. Morin, F. Fleurey, O. Barais, N. Plouzeau, and J.-M. Jézéquel. A Dynamic Component Model for Cyber Physical Systems. In CBSE 2012, Bertinoro, Italie, June 2012.
[10] R. Johnson and B. Woolf. The Type Object Pattern, 1997.
[11] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg. Models@ run.time to support dynamic adaptation. Computer, 42(10) :44–51, 2009.
[12] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani. A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. Software : Practice and Experience, 2011.
[13] J.-S. Sottet, V. Ganneau, G. Calvary, J. Coutaz, J.-M. Favre, and R. Demumieux. Model-driven adaptation for plastic user interfaces. In Proc. INTERACT 2007, the eleventh IFIP TC13 International Conference on Human-Computer Interaction, pages 397–410, 2007.